Matrox

Odyssey Developer's Toolkit

Development toolkit for native programming of the Motorola® G4 PowerPC™ microprocessor and Matrox Pixel Accelerator.

Harness the full power of the Matrox Odyssey family of vision processor boards

The Matrox Odyssey Developer's Toolkit (DTK) complements the Matrox Odyssey Software Development Toolkit (SDK) by providing additional programming levels. These levels allow developers to fully extract the power of the Matrox Odyssey family of vision processor boards. Developers can increase system determinism by running Matrox Imaging Library (MIL) and/or Matrox Odyssey Native Library (ONL) control code directly on the Motorola® G4 PowerPC™ microprocessor, further optimize an algorithm by merging ONL functions, and extend a MIL or ONL-based application with custom image processing and analysis functions implemented directly on the G4 or Matrox Oasis ASIC's Pixel Accelerator (PA).

Includes

- Matrox Odyssey Software Development Toolkit
  - Matrox Imaging Library (MIL) for Odyssey[1]
  - Matrox Odyssey Native Library (ONL)
- Matrox Odyssey Developer's Guide manual
- MIL for Odyssey and ONL in object code form (not linked)
- programming examples
- Matrox compiler and linker utilities
- advanced technical support

Requires

- Matrox Odyssey family vision processor board
- Metrowerks® CodeWarrior for PowerPC™ Embedded Systems and Motorola® documentation[2]

MATROX IMAGING

Preliminary

Increase system determinism

With the DTK, developers can easily move the control portion of an application from the host to the G4 to offload this time-critical task. As a result, real-time operations are guaranteed, even when the host CPU is running a non real-time OS and is busy with other system tasks (i.e., HMI, disk, network, I/O). Porting all or just a portion of the application from the host to the G4 requires no specialized knowledge of the G4. DTK users still have access to all MIL and ONL functions, and programming is performed in standard C/C++.

Optimize an ONL-based algorithm

DTK users can merge a sequence of ONL functions into a single new function to speed-up execution by reducing I/O with on-board main memory. Merged functions can run several times faster than the original sequence of ONL functions since intermediate results are stored in the G4's cache rather than in main memory. This kind of optimization is easy to implement since the DTK provides a large set of optimized low-level processing primitives that can be combined together to build more complex functions without writing any custom AltiVec™ code.

Using the DTK's low-level processing primitives

Consider an algorithm, which adds three 8-bit buffers (call them A, B, and C), then binarizes the result to an 8-bit value. The first implementation (shown below) uses standard ONL calls and a 16-bit temporary buffer to hold the intermediate results. The 16-bit temporary buffer is read twice and written twice, so the total number of memory accesses is 12 bytes/pixel instead of the ideal 4 bytes/pixel.

```
imIntDyadic(Thread, SrcA, SrcB, TmpBuf, IM_ADD, 0);
imIntDyadic(Thread, SrcC, TmpBuf, TmpBuf, IM_ADD, 0);
imIntBinarize(Thread, TmpBuf, Dst, IM_GREATER, Thresh, 0, 255, 0, 0);
```

The optimized version avoids all the unnecessary I/O by processing the image block-by-block using low-level processing primitives. It will execute almost three times faster than the first version, and doesn't require any custom AltiVec™ code. The main loop is written as follows.

```
for (Block = 0; Block < NumBlock; Block++)
{
  /* Read the first two sources and convert to 16-bit */
  NumPix = NextSrcA(&BlockSrcA, TmpBuf1);
  NextSrcB(&BlockSrcB, TmpBuf2);

  /* Add the first two sources */
  mp_1d_add_16(TmpBuf1, TmpBuf2, TmpBuf2, NumPix);

  /* Read the third source and convert to 16-bit */
  NextSrcC(&BlockSrcC, TmpBuf1);

  /* Add the third source */
  mp_1d_add_16(TmpBuf1, TmpBuf2, TmpBuf2, NumPix);

  /* Binarize the sum */
  mp_1d_bin_gt_u16(TmpBuf2, TmpBuf2, vThresh, v0, v255, v0,NumPix);

  /* Convert the result back to 8-bit and store it */
  NextDst(&BlockDst, TmpBuf2);
}
```

Write custom functions

While MIL and ONL offer an extensive list of functions, there are some cases where a developer will need to write custom G4 and/or PA code to perform a specialized operation. For this purpose, the DTK allows for programming of the G4's AltiVec™ unit and PA's processing elements directly.

Programming the G4 is done exclusively in C/C++ using language extensions and intrinsic functions. The DTK provides the framework to facilitate the writing of custom functions and dramatically reduces software development time. For example, when programming for the G4 directly, DTK users only need to write a simple inner loop to process whole aligned vectors. Variable image sizes, unaligned images, and mixed data types are all automatically supported. Moreover, memory bandwidth is maximized since pre-fetching is also handled automatically.

AltiVec™ programming

The first loop is standard C code, which adds two blocks of 8-bit pixels together (one pixel at a time).

```
unsigned char *Src1, *Src2, *Dst;
...
for (X = 0; X < SizeX; X++)
        Dst[X] = Src1[X] + Src2[X];
```

The next version uses the AltiVec™ programming model to perform the same operation much faster, because it works on one vector (sixteen 8-bit pixels) at a time.

```
vector unsigned char *vSrc1, *vSrc2, *vDst;
...
/* Get the size in vectors, not pixels */
SizeX = SizeX / vec_step(vector unsigned char);
for (X = 0; X < SizeX; X++)
        vDst[X] = vec_add(vSrc1[X], vSrc2[X]);
```

The only differences are that the pointers are declared to be a vector type, the loop counter is divided by 16 (because 16 pixels are processed at once), and the addition is specified by the AltiVec™ intrinsic function vec_add() rather than with the standard "+" operator.

Write custom functions (continued)

The PA is flexible enough to execute a wide variety of operations that may not have equivalents in MIL or ONL. The DTK makes it easy to do most of the PA set up, leaving you to implement only the nanocode to process each pixel. PA nanocode is rather like assembly code, but most custom functions only use short sequences of 5 to 10 instructions, so the effort required is minimal.

PA programming

Consider the algorithm described earlier (i.e., add three buffers and binarize the result). The following code sets up the PA to read the three source buffers and write the destination buffer.

```
/* Initialize for a point operation with three sources and one dest. */
mp_pa_set_mode(&PA, IM_OP_POINT, 3, 1, IM_SINGLE, IM_DISABLE);

/* Program the sources */
mp_pa_set_src(&PA, 1, SrcA, 0, SizeX, SizeY, IM_DEFAULT);
mp_pa_set_src(&PA, 2, SrcB, 0, SizeX, SizeY, IM_DEFAULT);
mp_pa_set_src(&PA, 3, SrcC, 0, SizeX, SizeY, IM_DEFAULT);

/* Program the destination */
mp_pa_set_dst(&PA, 0, Dst, 0, SizeX, SizeY, IM_DEFAULT);
```

The most complicated thing to do is writing the nanocode to process each pixel (see below), but in this case it consists of only four instructions. Note that the same code will execute simultaneously on all 64 processing elements of the PA.

```
; Inputs are:-
; R1        SrcA
; R2        SrcB
; R3        SrcC
; R4        0
; C0        Thresh
; C2        255
;
          ADD(R1, R2)     ; Add the first two buffers (result will be in A1)
          ADD(R3, A1)     ; Add the third to get the final sum
LOOP      SUB(C0, A1)     ; Thresh-Sum sets the N flag if Sum > Thresh
OUT       SEL(N, C2, R4)  ; Sum > Thresh ? 255 : 0
```

Development platform

The hardware platform used to create, compile and debug custom code is a host PC with a Matrox Odyssey family vision processor board. Code is generated on the host PC and then downloaded to the G4 for testing and debugging. Users will also need Metrowerks® CodeWarrior for PowerPC™ Embedded Systems. CodeWarrior consists of a compiler, linker and debugger all working within an integrated development environment. CodeWarrior makes embedded code development as easy as developing applications for the PC. Moreover, with CodeWarrior, there is no need for a hardware emulator since the debugger interacts directly with the vision processor board over the PCI/PCI-X bus.

Comes with preferred access

DTK customers have preferential access to our experienced staff of developers specializing in coding imaging functions for the Matrox Odyssey architecture.

Ordering Information

Part number	Description
Hardware	
MIL ONL WIN DTK	Matrox Odyssey Developer's Toolkit (DTK) under Windows®. Includes CD with MIL for Odyssey, ONL, MIL DTK and ONL DTK.

Users will also need to purchase Metrowerks® CodeWarrior for PowerPC™ Embedded Systems (version 6.1 or later) directly from Metrowerks®:

Software

Part number	Description
CE-EPPCx[3]	CodeWarrior for PowerPC™ Embedded Systems, hosted on Windows®

Notes:
1. MIL for Odyssey includes MIL for IA32 (host PC), which requires additional development or run-time license.
2. AltiVec™ Technology Programming Environments Manual, AltiVec™ Technology Programming Interface Manual and MPC7450 RISC Microprocessor Family User's Manual.
3. Where 'x' is version number.

Preliminary

matrox

For more information, please call: 1-800-804-6243 (toll free in North America)
or (514) 822-6020 or e-mail: imaging.info@matrox.com or http://www.matrox.com/imaging

Corporate headquarters:

Canada and U.S.A.
Matrox Electronic Systems Ltd.
1055 St. Regis Blvd.
Dorval, Quebec H9P 2T4
Canada
Tel: (514) 685-2630
Fax: (514) 822-6273

Offices:

Europe, Middle East & Africa
Matrox VITE Limited
Sefton Park
Stoke Poges
Buckinghamshire
SL2 4JS
U.K.
Tel: 01753 665511
Fax: 01753 665599

France
Matrox France SARL
2, rue de la Couture,
Silic 225
94528 Rungis Cedex
Tel: (0) 1 45-60-62-00
Fax: (0) 1 45-60-62-05

Germany
Matrox Electronic Systems GmbH
Inselkammerstr. 8
D-82008 Unterhaching
Germany
Tel: 089/62170-0
Fax: 089/614 9743